

TCP-NCL: A Unified Solution for TCP Packet Reordering and Random Loss

Chengdi Lai, Ka-Cheong Leung, and Victor O.K. Li
Department of Electrical and Electronic Engineering
The University of Hong Kong
Pokfulam Road, Hong Kong, China
E-mail: {laichengdi, kleung, vli}@eee.hku.hk

Abstract

In this paper, we propose a modified TCP variant, known as TCP for non-congestion loss (TCP-NCL), to tackle the problems of packet reordering and random packet loss in wireless networks. A packet retransmission decision model has been constructed to optimize the waiting time period before triggering an unacknowledged packet retransmission. Three algorithms, namely, TCP-NCL-RT, TCP-NCL-SRDT, and TCP-NCL-RTT-Update, have been developed within the framework of TCP-NCL. TCP-NCL-RT utilizes a retransmission timer to trigger a retransmission of a packet when its acknowledgment has not been received within a specified time period. TCP-NCL-SRDT employs a spurious retransmission detection timer, and is designed to reduce congestion control spuriously activated by packet reordering or random packet loss. TCP-NCL-RTT-Update is responsible for maintaining an accurate updated record of RTT throughout a TCP session. Our simulation studies reveal that the performance of TCP-NCL is robust to various types of packet reordering as well as random packet loss.

Index Terms

Transmission Control Protocol (TCP), wireless network, congestion control, flow control, packet reordering, spurious retransmission detection timer, random packet loss, retransmission timer, TCP-NCL, TCP-NCL-RT, TCP-NCL-SRDT

I. INTRODUCTION

Transmission Control Protocol (TCP) [14] is the most popular transport layer protocol over wired networks, providing in-order end-to-end data delivery service to various applications. With the promise of ubiquitous computation and information access, wireless network is drawing increasing research interest. Naturally, we need to adapt TCP to the wireless environment.

A. Challenges of TCP Design over Wireless Networks

Traditional TCP design relies on the assumptions of nearly in-order packet delivery and error-free transmission channel. Following these assumptions, any out-of-order packet events are treated as an indication of network overload, upon which congestion control measures [1], such as reducing the congestion window size, will be applied. However, in wireless networks, packet reordering can be persistent and packet loss can occur at a significant rate of up to 2% [13]. Consequently, the efficiency of a TCP sender will be seriously deteriorated in wireless networks since packet retransmission and congestion window reduction will be spuriously triggered from time to time, resulting in undesirable under-utilization of network bandwidth.

Packet reordering can be caused by a myriad of reasons. In [11], five major causes are listed: packet-level multi-path routing, route fluttering, inherent parallelism in modern high-speed routers, link-layer retransmission, and router forwarding lulls. In [10], two more scenarios commonly observed over mobile ad-hoc wireless networks are further identified: hand-off and path changes caused by topological changes. Based on their effects on the round trip time, we further classify these seven causes into two subsets: *Type I* and *Type II*. Their corresponding outcomes are termed *Type I Reordering* and *Type II Reordering*, respectively.

Type I Causes include inherent parallelism of routers, link-layer retransmission, and router forwarding lulls, which introduce minor yet persistent variations to the round trip time. This subset of scenarios constitute the dominant causes for packet reordering over fixed wireless networks.

Type II Causes cover all the remaining causes, which usually introduce significant increase or decrease in the round trip time. In mobile wireless networks, *Type II Reordering* is commonly observed. Yet, they take place much less frequently as compared with *Type I Reordering*. For a TCP sender employing one of the temporal approaches¹, an abrupt increase of the round trip time (a *Type II Reordering*) may undermine the accuracy in estimating the round trip time. The TCP sender would tend to trigger spurious retransmission when the round trip time is suddenly increased substantially and the corresponding timer expires before acknowledgments arrive. After that, the sender would be unable to tell without additional information whether the received acknowledgment is for the retransmitted packet or the original packet. It follows that the corresponding round trip time cannot be accurately estimated.

¹Temporal approaches are defined in [11] as a group of techniques which proactively defer congestion responses until the corresponding timer fires.

In contrast to the diverse causes for packet reordering, random packet loss is due solely to the inherent wireless channel error caused by signal fading, noise, and interference. Currently, some link layer retransmission mechanisms (LLRTX) [2], [8] have been proposed to tackle this problem. These mechanisms aim to make random packet loss transparent to the TCP layer by locally retransmitting the lost packets at the link layer, at the expense of altering the packet order of a flow. Nevertheless, LLRTX cannot guarantee zero random packet loss at the TCP layer. In high noise and high interference scenarios, the link layer may fail to transmit a certain packet in all the attempts allowed by the retransmission limit, thereby incurring non-congestion packet loss. Therefore, random packet loss remains an issue when TCP runs over wireless networks.

B. Related Work

The problems of packet reordering and random packet loss necessitate the modification of TCP to adapt it to wireless networks. Currently, these modified TCP variants generally fall into three different categories according to the problems they aim to tackle, namely, the problem of packet reordering, random packet loss, and both.

The solutions for packet reordering include RR-TCP [18], TCP-DCR [3], TCP-DOOR [16], and TCP-PR [4]. RR-TCP, TCP-DCR, and TCP-PR aim to deal with persistent packet reordering. Yet, under *Type II Reordering*, they offer distinctly different performance. As shown by our simulation results, the throughput of TCP-DCR can be significantly reduced when there is an abrupt increase in the round trip time (*Type II Reordering*). A spurious packet retransmission is triggered and the increased round trip time is ignored by its RTT sampling process. As a result, TCP-DCR will set its timer too conservatively, resulting in consecutive spurious activation of packet retransmission and congestion response. In contrast, TCP-PR can be made to be relatively robust to *Type II Reordering* by setting the expiration period of its timer in an aggressive manner, thus reducing the occurrence rate of spurious retransmission. RR-TCP improves the round trip time estimator. When a spurious retransmission leads to the arrival of two acknowledgments for the same packet, RR-TCP would sample the round trip time by computing the mean value of both RTT samples for that packet. The improvement is verified by our simulation to be effective in dealing with *Type II Reordering*.

TCP-DOOR is specifically designed to tackle *Type II Reordering*. It infers a path change upon the detection of an out-of-order event. Yet, as shown by our simulation results, persistent *Type I Reordering* can significantly undermine the performance of TCP-DOOR.

Additionally, all these TCP variants depend on LLRTX to deal with random packet loss and assume that all packet losses are due to network overload. As discussed previously, this assumption is unwarranted and can lead to performance deterioration in TCP under scenarios with high channel error rates.

The solutions for random packet loss include TCP-Veno [7] and TCP-Westwood (TCP-W) [5], which focus on differentiating between congestion loss and non-congestion loss while leaving packet reordering unattended. This limitation, however, restricts the applicability of TCP-Veno and TCP-W to networks with nearly in-order packet delivery. Particularly, their interoperability with LLRTX are undermined.

Efforts have also been made to develop a unified solution for both problems of packet reordering and random packet loss. ATCP [12] introduces an ATCP layer between TCP and IP, requiring major modifications to the network protocol stack. The ATCP layer switches TCP among various pre-defined states in accordance with the network condition, trying to avoid spurious packet retransmission and congestion response.

C. Our Contributions

The focus of this work is to propose a unified solution for the problems of packet reordering (both *Type I* and *Type II*) and random packet loss. A temporal approach, called TCP for non-congestion loss (TCP-NCL), is developed to tackle both problems within one compact framework composed of three algorithms: TCP-NCL-RT, TCP-NCL-SRDT, and TCP-NCL-RTT-Update. TCP-NCL-RT utilizes a retransmission timer (RT) to trigger a retransmission of a packet when its acknowledgment has not been received within a specified time period. TCP-NCL-SRDT employs a spurious retransmission detection timer (SRDT), and is designed to reduce congestion control spuriously activated by packet reordering or random packet loss. TCP-NCL-RTT-Update is responsible for maintaining an updated record of RTT throughout a TCP session, and performing reliable RTT sampling by taking *Type II Reordering* into consideration.

In order to determine an expiration period for RT (namely, the optimized waiting period for a packet retransmission), a packet retransmission decision (PRD) model is constructed. Within the PRD model, the concept of "potential cost" is introduced and potential costs of retransmission timeout (RTO) and spurious packet retransmission are computed, based on which an analytical expression for the optimal expiration period of RT is developed.

D. Organization of the Paper

This paper is organized as follows. Section II outlines the general behavior of TCP-NCL and describes the PRD model. Based on the PRD model, Section III discusses the TCP-NCL-RT, TCP-NCL-SRDT, and TCP-NCL-RTT-Update algorithms in detail. Section IV examines the performance of TCP-NCL against variations of its preset parameters and compares it with some popular TCP variants, namely, RR-TCP, TCP-DCR, TCP-DOOR, TCP-PR, TCP-Veno, and TCP-W. Section V concludes and discusses some possible extensions of our work.

II. PACKET RETRANSMISSION DECISION MODEL

In existing TCP variants, packet retransmission and congestion control mechanisms are generally bundled together. For example, RR-TCP retransmits a packet when the number of duplicate acknowledgments received reaches *dupthresh*. It then reduces *cwnd* and *ssthresh* once the retransmitted packet is acknowledged. TCP-DCR and TCP-PR retransmit an unacknowledged packet and activate congestion control simultaneously upon the expiration of a certain timer. However, with packet retransmission coupled with congestion control, a TCP sender encounters the dilemma of whether or not to retransmit an unacknowledged packet. If the sender activates *cwnd* packet retransmission and congestion response early, it will incur a high risk of spurious congestion response. Yet, by delaying packet retransmission until sometime later (when packet loss is almost certain), it will substantially increase the risk of RTO.

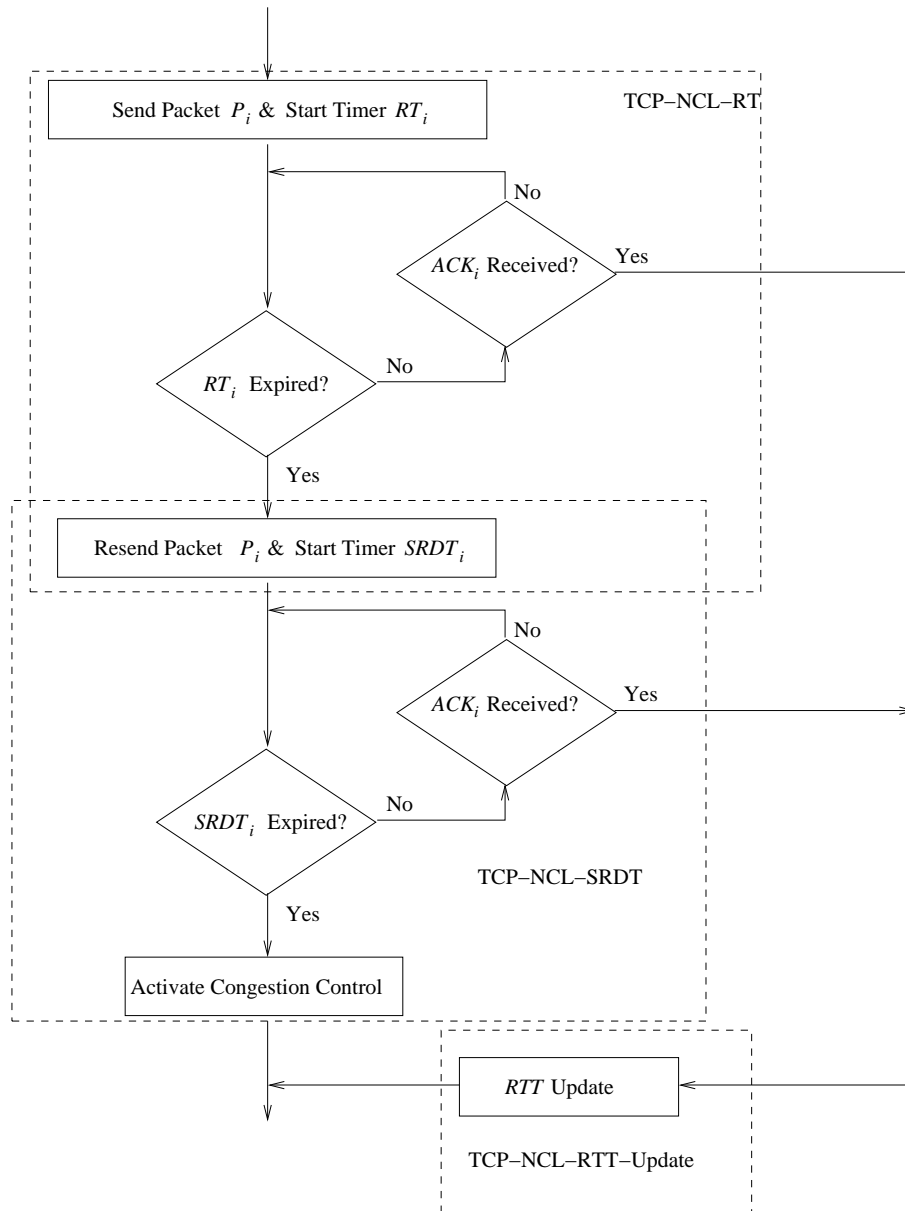


Fig. 1. A flowchart for TCP-NCL.

A. General Behavior of TCP-NCL

Based on the observation stated above, we decide to split packet retransmission and congestion control in our design, as illustrated in Fig. 1. A new retransmission timer RT_i is started whenever a new packet P_i is injected into the network. If ACK_i (an acknowledgment for P_i) is received at the TCP sender before RT_i expires, RT_i is cancelled. Otherwise, P_i will be retransmitted and a spurious retransmission detection timer $SRDT_i$ will be started.

$SRDT_i$ will be cancelled if ACK_i arrives before it expires. Otherwise, the congestion control mechanism will be activated upon the expiration of $SRDT_i$. In the activation of congestion control measures, we adopt TCP-PR's approach in that packet losses within the same congestion window will trigger the reduction of $cwnd$ and $ssthresh$ only once.

The rationale behind our design of TCP-NCL is as follows. Upon the expiration of RT_i , it is possible that P_i is dropped due to congestion or that P_i or ACK_i is still being transmitted or lost due to channel error. Therefore, we would retransmit P_i at this point to avoid the risk of RTO, and postpone the decision regarding whether to apply congestion control until $SRDT_i$ expires, when it is highly likely that P_i is lost due to network overload.

B. PRD Model

The PRD Model optimizes the waiting time before retransmitting an unacknowledged packet P_i , i.e. the expiration period of timer RT_i . From an analytical perspective, whenever we decide whether or not to retransmit an unacknowledged packet, the costs of RTO and spurious retransmission will have to be considered. Accordingly, the basic idea of PRD is to compare the potential cost (PO) of retransmitting P_i and that of delaying retransmitting P_i . An unacknowledged packet P_i shall be retransmitted once the former cost exceeds the latter.

To facilitate the subsequent derivation and discussion, we define the notations as shown in Fig. 2.

	Definition
AL	PL occurs to all other packets within the same window as P_i
C_A	Reduction of the total number of transmitted packets due to the occurrence of event A
$\bar{F}(t)$	Cumulative distribution function of RTT
$f(t)$	Probability density function of RTT
$P(A)$	Probability of event A occurring
\overline{PL}	P_i is lost or ACK_i is lost or $RTT > \tau_{RTO}$
\overline{PU}	Packet P_i is unacknowledged
\overline{SC}	Congestion response is spuriously activated
\overline{SR}	Packet P_i is spuriously retransmitted
\overline{TO}	RTO
σ	Size of the congestion window
σ_i	σ just before TO or SC
τ_T	Expiration period of timer T

Fig. 2. Definition of notations.

Let PO_{rtx} and PO_{drt} be the POs of retransmitting an unacknowledged packet P_i and delaying the retransmission of P_i , respectively. PO_{rtx} and PO_{drt} can be computed as:

$$\begin{aligned} PO_{rtx} &= P(SR|\overline{PU}) \cdot C_{SR} \\ &= P(\overline{PL}|\overline{PU}) \cdot C_{SR} \end{aligned} \quad (1)$$

$$PO_{drt} = P(\overline{TO}|\overline{PU}) \cdot C_{TO} \quad (2)$$

By defining PO in this manner, we treat throughput reduction as a cost and assumes that τ_{RT_i} takes on the optimal value when such cost is minimized. Yet, in the event of congestion loss, a timely activation of congestion control would be desirable and a reduction of TCP sender's throughput is necessary to avoid further overloading of the network. Therefore, in the subsequent derivation of PRD and the interpretation of the final outcome, congestion loss should be ignored. Nevertheless, as will be explained in Sec. III-B, this will not undermine TCP-NCL's responsiveness to congestion loss as compared with other TCP variants.

Initially, we have

$$P(\overline{PU}|\overline{PL}) = 1 - F(t) \quad (3)$$

$$P(\overline{PU}|PL) = 1 \quad (4)$$

In addition, we set

$$P(PL) = P_i \quad (5)$$

$$P(\overline{TO}|PL) = \epsilon_1 \quad (6)$$

$$F(\tau_{RT_i} + \tau_{SRDT_i}) = 1 - \epsilon_2 \quad (7)$$

By the law of total probability, we have the following:

$$\begin{aligned} P(TO) &= P(TO|PU) \cdot P(PU) + P(TO|\overline{PU}) \cdot P(\overline{PU}) \\ &= P(TO|PU) \cdot P(PU) \\ \Rightarrow P(TO|PU) &= \frac{P(TO)}{P(PU)} \end{aligned} \quad (8)$$

$$\begin{aligned} P(PL) &= P(PL|PU) \cdot P(PU) + P(PL|\overline{PU}) \cdot P(\overline{PU}) \\ &= P(PL|PU) \cdot P(PU) \\ \Rightarrow P(PL|PU) &= \frac{P(PL)}{P(PU)} \end{aligned} \quad (9)$$

$$\begin{aligned} P(TO) &= P(TO|PL) \cdot P(PL) + P(TO|\overline{PL}) \cdot P(\overline{PL}) \\ &= P(TO|PL) \cdot P(PL) \\ \Rightarrow P(TO|PL) &= \frac{P(TO)}{P(PL)} \end{aligned} \quad (10)$$

From (8), (9) and (10), we can further expand (2) as:

$$\begin{aligned} PO_{drt} &= \frac{P(TO)}{P(PU)} \cdot C_{TO} \\ &= \frac{P(TO)}{P(PL)} \cdot \frac{P(PL)}{P(PU)} \cdot C_{TO} \\ &= \epsilon_1 \cdot P(PL|PU) \cdot C_{TO} \end{aligned} \quad (11)$$

To obtain expressions for $P(\overline{PL}|PU)$ in (1) and $P(PL|PU)$ in (11), we apply Bayes' Theorem:

$$\begin{aligned} P(PL|PU) &= \frac{P(PL) \cdot P(PU|PL)}{P(PL) \cdot P(PU|PL) + P(\overline{PL}) \cdot P(PU|\overline{PL})} \\ &= \frac{P_l}{1 - F(t)(1 - P_l)} \end{aligned} \quad (12)$$

$$\Rightarrow P(\overline{PL}|PU) = 1 - \frac{P_l}{1 - F(t)(1 - P_l)} \quad (13)$$

Now the remaining unknowns are C_{SR} in (1) and C_{TO} in (11), the expressions for which we derive next. When an RTO occurs and σ_i is larger than 1, σ will be reset to 1 and $ssthresh$ will be set $0.5\sigma_i$. Here we assume that ACK arrives before the occurrence of another RTO timeout. Thus, after being reset, the TCP sender is at the slow start stage and σ will double itself at every round trip time as long as it is less than $ssthresh$, which is $0.5\sigma_i$. Once σ reaches $\lceil ssthresh \rceil$, the TCP sender will leave the slow start stage and enter the congestion avoidance stage. Thus, let n_1 be the number of cycles during which the TCP sender is in the slow start stage, we will have:

$$\begin{aligned} 2^{n_1} &\geq 0.5\sigma_i \\ 2^{n_1-1} &< 0.5\sigma_i \\ \Rightarrow n_1 &= \lceil \log_2 \sigma_i \rceil - 1 \end{aligned} \quad (14)$$

At the congestion avoidance stage, σ will be incremented by 1 at every cycle. Let n_2 be the number of round trip cycles during which TCP sender is in the congestion avoidance stage with a σ less than σ_i . At the first RTT cycle of the slow start stage, $\sigma = \lceil 0.5\sigma_i \rceil$. At the n_2 'th cycle, $\sigma = \sigma_i - 1$. Thus, we have

$$\begin{aligned} n_2 + \lceil 0.5\sigma_i \rceil - 1 &= \sigma_i - 1 \\ \Rightarrow n_2 &= \sigma_i - \lceil 0.5\sigma_i \rceil \end{aligned} \quad (15)$$

Thus, we obtain C_{TO} as:

$$\begin{aligned} C_{TO} &= \sigma_i(n_1 + n_2) - \left[\sum_{i=1}^{n_1} 2^{i-1} + \sum_{i=1}^{n_2} (\lceil 0.5\sigma_i \rceil + i - 1) \right] \\ &= 0.5\sigma_i^2 + (-\lceil 0.5\sigma_i \rceil + \lceil \log_2 \sigma_i \rceil - 0.5)\sigma_i + (0.5\lceil 0.5\sigma_i \rceil^2 - 0.5\lceil 0.5\sigma_i \rceil - 0.5 \times 2^{\lceil \log_2 \sigma_i \rceil} + 1) \\ \Rightarrow C_{TO} &= \begin{cases} 0.5\sigma_i^2 + (-\lceil 0.5\sigma_i \rceil + \lceil \log_2 \sigma_i \rceil - 0.5)\sigma_i + (0.5\lceil 0.5\sigma_i \rceil^2 - 0.5\lceil 0.5\sigma_i \rceil - 0.5 \times 2^{\lceil \log_2 \sigma_i \rceil} + 1) & (\sigma_i \geq 2) \\ 0, & (\sigma_i = 1) \end{cases} \end{aligned} \quad (16)$$

To obtain C_{SR} , losses introduced by spurious packet retransmission and spurious congestion response should both be considered. When a spurious retransmission is detected, we would reduce the number of packets to be transmitted in the current round trip cycle by one so as to strictly observe the limitation of the congestion window. This reduction should be counted as part of C_{SR} . Thus

$$C'_{SR} = 1 \quad (17)$$

On the other hand, when an unacknowledged packet is retransmitted untimely and σ_i is larger than 1, the risk of spurious congestion response needs to be considered as well. Thus PO of spurious congestion response, PO_{scp} , is introduced for this purpose and defined as:

$$\begin{aligned} C''_{SR} &= PO_{scp} \\ &= P(SC|PU) \cdot C_{SC} \quad (\sigma_i \geq 2) \end{aligned} \quad (18)$$

When packet P_i is lost due to random channel error and is retransmitted, spurious congestion response can be avoided if and only if ACK_i triggered by the retransmitted packet arrives at the TCP sender before $SRDT_i$ expires. However, as will be discussed later, τ_{SRDT_i} is set artificially and there may be insufficient time for the retransmitted packet to be acknowledged. Thus, to strengthen the robustness of TCP-NCL to the variation of τ_{SRDT_i} , and to simplify later computations, here we consider the worst case only, i.e. ACK_i for the retransmitted P_i always fails to arrive before $SRDT_i$ expires. Thus,

$$P(SC|PL) = 1 \quad (19)$$

When packet P_i is not lost and is spuriously retransmitted, spurious congestion response can be avoided if the acknowledgment for either the original P_i or the retransmitted P_i (denoted as P'_i to avoid confusion) arrives at the TCP sender before $SRDT_i$ expires. By the same token as above, we assume the retransmitted P_i cannot be acknowledged in time. Thus, we have:

$$\begin{aligned} P(\overline{SC}|\overline{PL}) &= F(t + \tau_{SRDT_i}) \\ \Rightarrow P(SC|\overline{PL}) &= 1 - F(t + \tau_{SRDT_i}) \end{aligned} \quad (20)$$

By the law of total probability, we have:

$$\begin{aligned} P(SC) &= P(SC|PL) \cdot P(PL) + P(SC|\overline{PL}) \cdot P(\overline{PL}) \\ &= 1 - F(t + \tau_{SRDT_i})(1 - P_l) \\ P(PU) &= P(PU|PL) \cdot P(PL) + P(PU|\overline{PL}) \cdot P(\overline{PL}) \\ &= 1 - F(t)(1 - P_l) \\ P(SC) &= P(SC|PU) \cdot P(PU) + P(SC|\overline{PU}) \cdot P(\overline{PU}) \\ &= P(SC|PU) \cdot P(PU) \\ \Rightarrow P(SC|PU) &= \frac{P(SC)}{P(PU)} \\ &= \frac{1 - F(t + \tau_{SRDT_i})(1 - P_l)}{1 - F(t)(1 - P_l)} \\ \Rightarrow C''_{SR} &= \frac{1 - F(t + \tau_{SRDT_i})(1 - P_l)}{1 - F(t)(1 - P_l)} \cdot C_{SC} \quad (\sigma_i \geq 2) \end{aligned} \quad (21)$$

A lower bound for C''_{SR} can be obtained as:

$$C''_{SR} \geq \frac{1 - F(t + \tau_{SRDT_i})}{1 - F(t)} \cdot C_{SC} \quad (22)$$

Upon the activation of congestion response, both σ and $ssthresh$ will be set to $0.5 \sigma_i$. Thus TCP sender is at the congestion avoidance stage to begin with. Let n_3 be the number of cycles during which TCP sender has a σ less than σ_i . Accordingly, we have

$$\begin{aligned} n_3 + \lceil 0.5 \sigma_i \rceil - 1 &= \sigma_i - 1 \\ \Rightarrow n_3 &= \sigma_i - \lceil 0.5 \sigma_i \rceil \\ \Rightarrow C_{SC} &= \sigma_i \cdot n_3 - \sum_{i=1}^{n_3} (\lceil 0.5 \sigma_i \rceil + i - 1) \\ &= (\sigma_i - \lceil 0.5 \sigma_i \rceil) \cdot (0.5 \sigma_i - 0.5 \lceil 0.5 \sigma_i \rceil + 0.5) \end{aligned} \quad (23)$$

Substitute (23) into (21),

$$C''_{SR} = \frac{1 - F(t + \tau_{SRDT_i})(1 - P_l)}{1 - F(t)(1 - P_l)} \cdot (\sigma_i - [0.5 \sigma_i]) \cdot (0.5 \sigma_i - 0.5 [0.5 \sigma_i] + 0.5) \quad (\sigma_i \geq 2) \quad (24)$$

$$\begin{aligned} \Rightarrow C_{SR} &= C'_{SR} + C''_{SR} \\ &= 1 + \frac{1 - F(t + \tau_{SRDT_i})(1 - P_l)}{1 - F(t)(1 - P_l)} \cdot (\sigma_i - [0.5 \sigma_i]) \cdot (0.5 \sigma_i - 0.5 [0.5 \sigma_i] + 0.5) \quad (\sigma_i \geq 2) \end{aligned} \quad (25)$$

$$\Rightarrow C_{SR} = \begin{cases} 1 + \frac{1 - F(t + \tau_{SRDT_i})(1 - P_l)}{1 - F(t)(1 - P_l)} \cdot (\sigma_i - [0.5 \sigma_i]) \cdot (0.5 \sigma_i - 0.5 [0.5 \sigma_i] + 0.5), & (\sigma_i \geq 2) \\ 1, & (\sigma_i = 1) \end{cases} \quad (26)$$

Substitute (6), (12), (13), (16) and (26) into (1) and (11), we have

$$PO_{rtx} = \begin{cases} \left[1 + \frac{1 - F(t + \tau_{SRDT_i})(1 - P_l)}{1 - F(t)(1 - P_l)} \cdot L_1(\sigma_i) \right] \left(1 - \frac{P_l}{1 - F(t)(1 - P_l)} \right), & (\sigma_i \geq 2) \\ 1 - \frac{P_l}{1 - F(t)(1 - P_l)}, & (\sigma_i = 1) \end{cases} \quad (27)$$

$$PO_{drt} = \begin{cases} L_2(\sigma_i) \cdot \frac{\epsilon_1 \cdot P_l}{1 - F(t)(1 - P_l)}, & (\sigma_i \geq 2) \\ 0, & (\sigma_i = 1) \end{cases} \quad (28)$$

where

$$L_1(\sigma_i) = 0.5 \sigma_i^2 + (-[0.5 \sigma_i] + 0.5) \sigma_i + (0.5 [0.5 \sigma_i]^2 - 0.5 [0.5 \sigma_i]) \quad (29)$$

$$L_2(\sigma_i) = 0.5 \sigma_i^2 + (-[0.5 \sigma_i] + [\log_2 \sigma_i] - 0.5) \sigma_i + (0.5 [0.5 \sigma_i]^2 - 0.5 [0.5 \sigma_i] - 0.5 \times 2^{\lceil \log_2 \sigma_i \rceil} + 1) \quad (30)$$

From (22), we can obtain a lower bound for PO_{rtx} as:

$$PO'_{rtx} = \begin{cases} \left[1 + \frac{1 - F(t + \tau_{SRDT_i})}{1 - F(t)} \cdot L_1(\sigma_i) \right] \left(1 - \frac{P_l}{1 - F(t)(1 - P_l)} \right), & (\sigma_i \geq 2) \\ 1 - \frac{P_l}{1 - F(t)(1 - P_l)}, & (\sigma_i = 1) \end{cases} \quad (31)$$

The optimal value of τ_{RT_i} is obtained when $PO_{rtx} = PO_{drt}$.

Let $POd = PO_{rtx} - PO_{drt}$, $POd' = PO'_{rtx} - PO_{drt}$, we have:

$$POd \geq POd' \quad (32)$$

Set $POd = 0$, the value of t thus obtained will be τ_{RT_i} . According to (32), for $t = \tau_{RT_i}$,

$$\begin{aligned} POd' &\leq POd = 0 \\ \Rightarrow PO'_{rtx} &\leq PO_{drt} \end{aligned} \quad (33)$$

Substitute (7), (28) and (31) into (33), we can get:

$$\begin{aligned} \left[1 + \frac{\epsilon_2}{1 - F(\tau_{RT_i})} \cdot L_1(\sigma_i) \right] \left(1 - \frac{P_l}{1 - F(\tau_{RT_i})(1 - P_l)} \right) &\leq L_2(\sigma_i) \cdot \frac{\epsilon_1 \cdot P_l}{1 - F(\tau_{RT_i})(1 - P_l)} \\ \Rightarrow \left[1 + \frac{\epsilon_2}{1 - F(\tau_{RT_i})} \cdot L_1(\sigma_i) \right] (1 - P_l) [1 - F(\tau_{RT_i})] &\leq L_2(\sigma_i) \cdot \epsilon_1 \cdot P_l \\ \Rightarrow [1 - F(\tau_{RT_i})]^2 + \epsilon_2 L_1(\sigma_i) [1 - F(\tau_{RT_i})] &\leq L_2(\sigma_i) \cdot \epsilon_1 \cdot \frac{P_l}{1 - P_l} \cdot [1 - F(\tau_{RT_i})] \\ \Rightarrow [1 - F(\tau_{RT_i})] \left[1 + \epsilon_2 \cdot L_1(\sigma_i) - \frac{P_l}{1 - P_l} \cdot \epsilon_1 \cdot L_2(\sigma_i) - F(\tau_{RT_i}) \right] &\leq 0 \end{aligned} \quad (34)$$

Therefore, for $\sigma_i \geq 2$, we have

$$F(\tau_{RT_i}) \begin{cases} \geq 1 + \epsilon_2 \cdot L_1(\sigma_i) - \frac{P_l}{1 - P_l} \cdot \epsilon_1 \cdot L_2(\sigma_i), & \epsilon_2 \cdot L_1(\sigma_i) < \frac{P_l}{1 - P_l} \cdot \epsilon_1 \cdot L_2(\sigma_i) \\ = 1, & \epsilon_2 \cdot L_1(\sigma_i) \geq \frac{P_l}{1 - P_l} \cdot \epsilon_1 \cdot L_2(\sigma_i) \end{cases} \quad (35)$$

III. TCP-NCL ALGORITHM

PRD provides an optimal value for τ_{RT_i} in the form of $F(\tau_{RT_i}) = C$. Naturally we are motivated to obtain the actual value of τ_{RT_i} by inverting $F(t)$. One approach is to construct a distribution model for RTT and subsequently develop an analytical expression of $F(t)$. The advantage of doing so is that there is no need to store the RTT samples once the distribution is formed. Unfortunately, such an RTT distribution is hard to determine. We leave this analytical approach as part of the future work. For now, we choose to record RTT samples to facilitate determining the value of τ_{RT_i} .

TCP-NCL is composed of three algorithms as indicated in Fig. 1: TCP-NCL-RT for packet retransmission, TCP-NCL-SRDT for spurious retransmission detection and congestion control, and TCP-NCL-RTT-Update for RTT sampling. The algorithms will be discussed next. The complete TCP-NCL framework is presented in Fig. 3.

A. TCP-NCL-RT

In Sec. II, we have obtained the optimal value for $F(\tau_{RT_i})$ in (35). In this section, we aim to further simplify this result so as to facilitate actual implementation.

In either of the two cases listed in (35), the following inequality will hold

$$F(\tau_{RT_i}) \geq 1 - \frac{P_l}{1 - P_l} \cdot \epsilon_1 \cdot L_2(\sigma_i) \quad (36)$$

where $\epsilon_1 = P(TO|PL)$ according to (6).

In TCP-NCL, the RTO timer is implemented on a per-flow basis. The expiration period of the RTO timer, τ_{RTO} , is initially set as:

$$\tau_{RTO_i} = \tau_{RT_i} + \tau_{SRDT_i} \quad (37)$$

For a lost packet P_i , if new non-duplicated acknowledgments arrive after its original transmission, the RTO timer will get reset and τ_{RTO} will be effectively extended. Otherwise, (37) will be the final value for τ_{RTO} . Therefore, we observe that (37) serves as a lower bound for the final value of τ_{RTO} .

If packet P_i is retransmitted at time τ_{RT_i} after its original transmission, RTO can be avoided if and only if this retransmitted packet can be acknowledged before the expiration time of the RTO timer. Therefore

$$\begin{aligned} \epsilon_1 &= P(TO|PL) \\ &= 1 - F(\tau_{RTO} - \tau_{RT_i}) \\ &\leq 1 - F(\tau_{RTO_i} - \tau_{RT_i}) \\ &= 1 - F(\tau_{SRDT_i}) \end{aligned} \quad (38)$$

Under non-congestion loss, there is a good chance that the retransmitted packets will get acknowledged in time to prevent RTO (i.e. $F(\tau_{SRDT_i})$ is closer to 1). Thus, ϵ_1 can be negligibly small. Under such cases, the optimal choice for the value of $F(\tau_{RT_i})$ can be established by the following two lemmas.

Lemma 1: $\lim_{\epsilon_1 \rightarrow 0} F(\tau_{RT_i}) = 1$.

Proof:

Taking limit on both sides in (36),

$$\lim_{\epsilon_1 \rightarrow 0} F(\tau_{RT_i}) \geq \lim_{\epsilon_1 \rightarrow 0} [1 - \frac{P_l}{1 - P_l} \cdot \epsilon_1 \cdot L_2(\sigma_i)] = 1 \quad (39)$$

Since $F(\tau_{RT_i}) \leq 1$,

$$\lim_{\epsilon_1 \rightarrow 0} F(\tau_{RT_i}) = 1 \quad (40)$$

Lemma 1 reveals that $F(\tau_{RT_i})$ gets close to 1 for a sufficiently small ϵ_1 . Intuitively, we assume that values for $F(\tau_{RT_i})$ in the interval [0.9,1] can be approximated as 1. Thus, we are interested in the range of values for ϵ_1 that will establish $F(\tau_{RT_i}) \geq 0.9$.

Lemma 2: $F(\tau_{RT_i}) \geq 0.9$ if $\epsilon_1 \leq \frac{0.1(1 - P_l)}{P_l L_2(\sigma_i)}$.

Proof:

$$\begin{aligned} F(\tau_{RT_i}) &\geq 1 - \frac{P_l}{1 - P_l} \cdot \epsilon_1 \cdot L_2(\sigma_i) \\ &\geq 1 - \frac{P_l}{1 - P_l} \cdot \frac{0.1(1 - P_l)}{P_l L_2(\sigma_i)} \cdot L_2(\sigma_i) \\ &= 0.9 \end{aligned} \quad (41)$$

Typically, $P_l \leq 0.15$. Consequently, the constraint of Lemma 2 would dictate that $\epsilon_1 \leq 0.56 L_2^{-1}(\sigma_i)$. The table below shows the upper bounds of ϵ_1 corresponding to a range of different values for σ_i .

σ_i	2	4	8	16	32	64	128
$max(\epsilon_1)$	0.5667	0.0944	0.0246	0.0074	0.0023	0.0007	0.0002

Hence, in the case of non-congestion loss, it should be justifiable to have the approximation

$$\begin{aligned} F(\tau_{RT_i}) &\simeq 1 \\ \Rightarrow \tau_{RT_i} &\simeq max(RTT) \end{aligned} \quad (42)$$

In the case of congestion loss, this assignment of τ_{RT_i} does not correspond to an optimal solution. Nevertheless, as will be shown later, the responsiveness of TCP-NCL to network overloading is still competitive as compared with other TCP variants.

B. TCP-NCL-SRDT

In Fig. 1, when ACK_i arrives at the sender during the period between the expiration of RT_i and $SRDT_i$, known as the *RT-SRDT interval*, congestion control will not apply. This ACK_i can be due to the originally transmitted or the retransmitted packet P_i . Our intention here is for the timer $SRDT_i$ to assume a sufficiently small expiration period so that either one of the following two cases will hold:

- 1) If the originally transmitted P_i triggers this ACK_i , there is no need to reduce *cwnd* or *thresh* since no packet loss occurs.
- 2) If the retransmitted packet triggers the received ACK_i , we can infer that RTT is too short. This in turn indicates a very small chance of network overload. As for the originally transmitted packet, it can either be reordered or lost due to random channel error. In either event, it would be unnecessary to activate congestion control mechanisms.

Therefore, a TCP sender will decide that congestion loss has occurred and congestion control mechanisms should come into play only when $SRDT_i$ has expired, and by then both packet reordering and random packet loss would be very unlikely.

Intuitively, we set τ_{SRDT_i} as:

$$\tau_{SRDT_i} = min(RTT) + \Delta T \quad (43)$$

Here, $min(RTT)$ denotes the minimum value among the recorded RTT samples. Obviously, it qualifies for our prescribed "sufficiently small value". The term ΔT is introduced for minor adjustment. By increasing ΔT , a TCP sender can detect spurious retransmission more effectively. In our actual implementation, because timestamps are utilized for realizing SRDT timers and a minor delay in activating congestion control will be inherently introduced, we set ΔT to zero to avoid an excessively aggressive TCP sender.

At this point, it can also be noticed that the delay between when the original packet is sent and when the congestion control mechanisms are triggered can be computed as:

$$\begin{aligned} Delay &= max(RTT) + min(RTT) + \Delta T \\ &\simeq 2 srtt + \Delta T \end{aligned} \quad (44)$$

where *srtt* denotes the average RTT.

In other temporal approaches such as TCP-DCR and TCP-PR, the TCP sender also delays congestion control for about $2 srtt$ or longer. Therefore, although our major focus is devoted to packet reordering and random packet loss, TCP-NCL is nevertheless competent in terms of responsiveness to congestion loss.

C. TCP-NCL-RTT-Update

The distribution of RTT, $F(t)$, is time-variant over most wireless networks. For example, RTT can be approximated as a linear function of *cwnd* over ad hoc wireless networks [15]. Since *cwnd* evolves with time, $F(t)$ will also vary over the entire TCP session. The time-variance property of $F(t)$ requires periodic updating of RTT samples, so that recent RTT samples are recorded and some outdated samples are discarded. As shown in Fig. 3, we would first define the *maximum RTT record length* (MRRL). Whenever the total number of stored RTT samples exceeds MRRL, the oldest samples will be discarded.

As can be observed in Fig. 1, the RTT record will be updated if ACK_i arrives before the expiration of RT_i or in the *RT-SRDT interval*. In the latter case, there is an ambiguity regarding whether the ACK_i is for the originally transmitted packet or the retransmitted packet. Thus, some further measures would be needed to make sure that ACK_i is for the original P_i before recording the corresponding RTT sample. For this purpose, we measure the time elapsed between the retransmission of P_i and the arrival of ACK_i , *delay-after-rtx*. If the measured value is less than $\beta \tau_{SRDT_i}$ ($0 < \beta < 1$), we infer that the retransmitted P_i cannot be acknowledged within such a short interval and thus record the RTT sample accordingly. Otherwise, the corresponding RTT sample will be ignored.

Algorithm	Description
TCP-NCL-RT	<pre> Transmit (P_i) Start$RT_i(maxRTT)$ </pre>
TCP-NCL-SRDT	<pre> if status (RT_i) = EXPIRED then { Retransmit (P_i) Start$SRDT_i(minRTT + \Delta T)$ } </pre>
TCP-NCL-RTT-Update	<pre> Procedure NewAck (ACK_i) { // called when a new ACK arrives if status (RT_i) = PENDING then { UpdateRTT ($RTTSample$) Cancel (RT_i) } else if status($SRDT_i$) = PENDING then { if delay-after-rtx < $\beta \tau_{SRDT_i}$ then UpdateRTT ($RTTSample$) Cancel ($SRDT_i$) } } Procedure UpdateRTT ($RTTSample$) { RTTRecord.Add($RTTSample$) While Size($RTTRecord$) > MRRL Do Discard ($RTTSample$) maxRTT \leftarrow max ($RTTRecord$) minRTT \leftarrow min ($RTTRecord$) } </pre>

Fig. 3. The complete TCP-NCL framework.

By updating RTT based on acknowledgments received in the RT - $SRDT$ interval, we try to increase the robustness of the RTT sampling process to changes in the network environment, especially to the occurrence of *Type II Reordering*. When there is an abrupt increase in RTT, it is likely that ACK_i will not be received before RT_i expires. Yet, the corresponding RTT sample can still be accurately recorded as long as it does not exceed $(\tau_{RT_i} + \beta \tau_{SRDT_i})$.

Both MRRL and β need to be determined with care. MRRL should be large enough to allow the storage of a sufficiently large RTT sample, but should also be appropriately limited so that outdated samples can be discarded in time. β should be sufficiently small to prevent the acknowledgments for retransmitted packets from affecting the RTT sampling process. Yet, a too conservative β will undermine the robustness of TCP-NCL-RTT-Update to *Type II Reordering*. Nevertheless, as will be shown by our simulation results, the performance of TCP-NCL is robust over a wide range of combinations of values for MRRL and β . An analytical approach for optimizing β and MRRL is left as part of the future work.

IV. PERFORMANCE EVALUATION

In this section, we present our simulation results. Section IV-A explains the simulation setup. Section IV-B examines the performance of TCP-NCL against various combinations of values for MRRL and β . Section IV-C compares the performance of TCP-NCL with some popular TCP variants, namely, RR-TCP, TCP-DCR, TCP-DOOR, TCP-PR, TCP-Veno, and TCP-W.

A. Simulation Setup

Three different network topologies, as illustrated in Fig. 4, have been used in the simulation experiment: an infrastructure-based wireless network, a multi-hop wireless network, and a wired network with a bottleneck link. We examine the performance of TCP variants under random packet loss, *Type I Reordering*, and *Type II Reordering*, respectively.

In the infrastructure-based wireless network, a TCP sender and a TCP receiver are connected through some wired and wireless links. Random channel error from 0 to 15% is deliberately introduced into each wireless link. The link layer retransmission mechanism is disabled to simulate an orderly TCP flow.

In the multi-hop wireless network, a TCP sender is connected to a TCP receiver via four wireless links. Random channel error is introduced into the wireless links with an error rate ranging from 0 to 15%. Link layer retransmission is enabled to introduce persistent packet reordering (*Type I Reordering*). Under high channel error rate, however, local link layer retransmission cannot guarantee successful packet delivery due to the retransmission limit (set to 3 in this case). Consequently, TCP will be confronted with both *Type I Reordering* and random packet loss.

In the wired network, the TCP connection traverses through a bottleneck link. The link delay of the bottleneck link takes on a random value from the interval $[50, maxDy]$ ms and is changed every 20 seconds. $maxDy$ ranges from 100 ms to 700 ms. This way, the round trip time will vary abruptly yet infrequently, as an approximation of *Type II Reordering*.

Two sets of simulation experiments have been carried out over the three network topologies described above. Their simulation results are presented in Sec. IV-B and Sec. IV-C, respectively. The first set of experiments examine the performance of TCP-NCL against various combinations of values for its two preset parameters, namely, MRRL and β . MRRL takes on a value from the set $\{250, 300, 400, 500, 600, 700, 800, 900, 1000\}$. β takes on a value from the set $\{0.3, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$. All possible combinations of the two sets of values have been covered by our tests.

The second set of experiments compare the performance of TCP-NCL with some popular TCP variants, namely, RR-TCP, TCP-DCR, TCP-DOOR, TCP-PR, TCP-Veno, and TCP-W. All the tests in this set assign MRRL as 1000 and β as 0.8 for TCP-NCL senders.

All the simulation experiments have been run over Network Simulator (ns) Version 2.29 [6]. In each test, a total of 20 runs have been performed to compute an average value and a 95% confidence interval of the performance metric, i.e. goodput in megabit per second (Mbps).

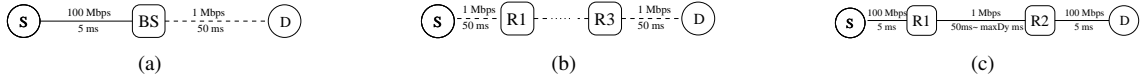


Fig. 4. The network topologies used for performance comparison: (a) Infrastructure-based wireless network. (b) Multi-hop wireless network. (c) Wired network with a bottleneck link.

B. Parameter Tuning

The simulation results over the infrastructure-based wireless network are shown in Fig. 5. When β is set to be 1, TCP-NCL suffers serious performance deterioration as channel error rate increases. This is because a TCP-NCL sender with $\beta = 1$ will treat an acknowledgment received before $SRDT_i$ expires as being triggered by the originally transmitted packet and record the RTT sample accordingly. Yet, non-congestion loss is the dominant reason for packet loss over the current network topology. Thus, acknowledgments for the retransmitted packets are highly likely to arrive at the TCP sender before $SRDT_i$ expires. When TCP-NCL-RTT-Update fails to ignore these acknowledgments, the accuracy of the RTT samples would be compromised.

Nevertheless, our simulation results also reveal that significant performance improvement can be immediately achieved with β being set to a value less than 1. When β varies between 0.3 and 0.95, TCP-NCL offers consistently high connection goodput regardless the value for MRRL and the channel error rate.

The simulation results over the multi-hop wireless network are shown in Fig. 6. TCP-NCL essentially maintains identical goodput performance with MRRL and β varying within their correspondingly prescribed sets of values. Therefore, over the multi-hop wireless network, TCP-NCL demonstrates very strong robustness to variations of its preset parameters.

The simulation results over the wired network are shown in Fig. 7. For all values of MRRL, TCP-NCL senders perform slightly worse when $\beta = 1$ as compared with their counter parts with a smaller β . This should be due to the occasional mistakes in differentiating between acknowledgments for the originally transmitted packets and the retransmitted packets.

On the other hand, the value of MRRL begins to affect the performance of TCP-NCL this time. When MRRL is set to 300 or a larger value and β is less than one, TCP-NCL renders approximately the same performance.

When MRRL is set to 250 and β varies between 0.5 and 0.95, TCP-NCL attains the best connection goodput among all the tests over the wired network. As discussed previously in Sec. IV-A, the wired network topology aims to simulate periodic occurrence of *Type II Reordering*. Whenever *Type II Reordering* takes place, it would be desirable for a TCP sender to replace all the RTT samples in storage with new samples as soon as possible. Apparently, a TCP-NCL sender with a small MRRL are at an advantage of attaining this.

Yet, when β is further reduced to 0.3, the goodput of TCP-NCL drops despite the merit of the small value of MRRL. This is because the excessively conservative value of β prevents TCP-NCL from recording RTT samples when there is an abrupt increase in RTT. Consequently, both RT_i and $SRDT_i$ will not be able to adjust their expiration periods accordingly, thereby triggering spurious packet retransmission and congestion response.

C. Performance Comparison

The simulation results for performance comparison are shown in Fig. 8. In the infrastructure-based wireless network, TCP-NCL essentially maintain a stable goodput level against channel error rates from 0 to 15%, whereas almost all of the other TCP variants experience drastic goodput decrease as the error rate increases. The only exception is TCP-W, which exhibits a relatively elegant performance deterioration. The performance of TCP-NCL should be attributable to the effectiveness of TCP-NCL-SRDT in differentiating between congestion loss and random packet loss. In contrast, RR-TCP, TCP-DCR, TCP-DOOR, and TCP-PR excludes the possibility of random packet loss, resulting in under-utilization of network resources.

In the multi-hop wireless network, TCP-DCR, TCP-NCL, and TCP-PR outperform other variants under channel error rate less than 9%, thereby demonstrating robustness to *Type I Reordering*. When the error rate further increases and random packet loss coexists with *Type I Reordering*, TCP-NCL performs slightly better than TCP-PR while the performance of TCP-DCR is seriously deteriorated. Again, in the latter scenario, TCP-NCL-SRDT plays a crucial role in achieving performance improvement for TCP-NCL.

In the wired network topology simulating *Type II Reordering*, RR-TCP, TCP-NCL, and TCP-PR offer the best connection goodput. For temporal approaches (TCP-DCR, TCP-NCL, and TCP-PR), the key to a good performance under *Type II Reordering* is an accurate sampling of RTT, based on which TCP differentiates packet reordering from packet loss. Thus,

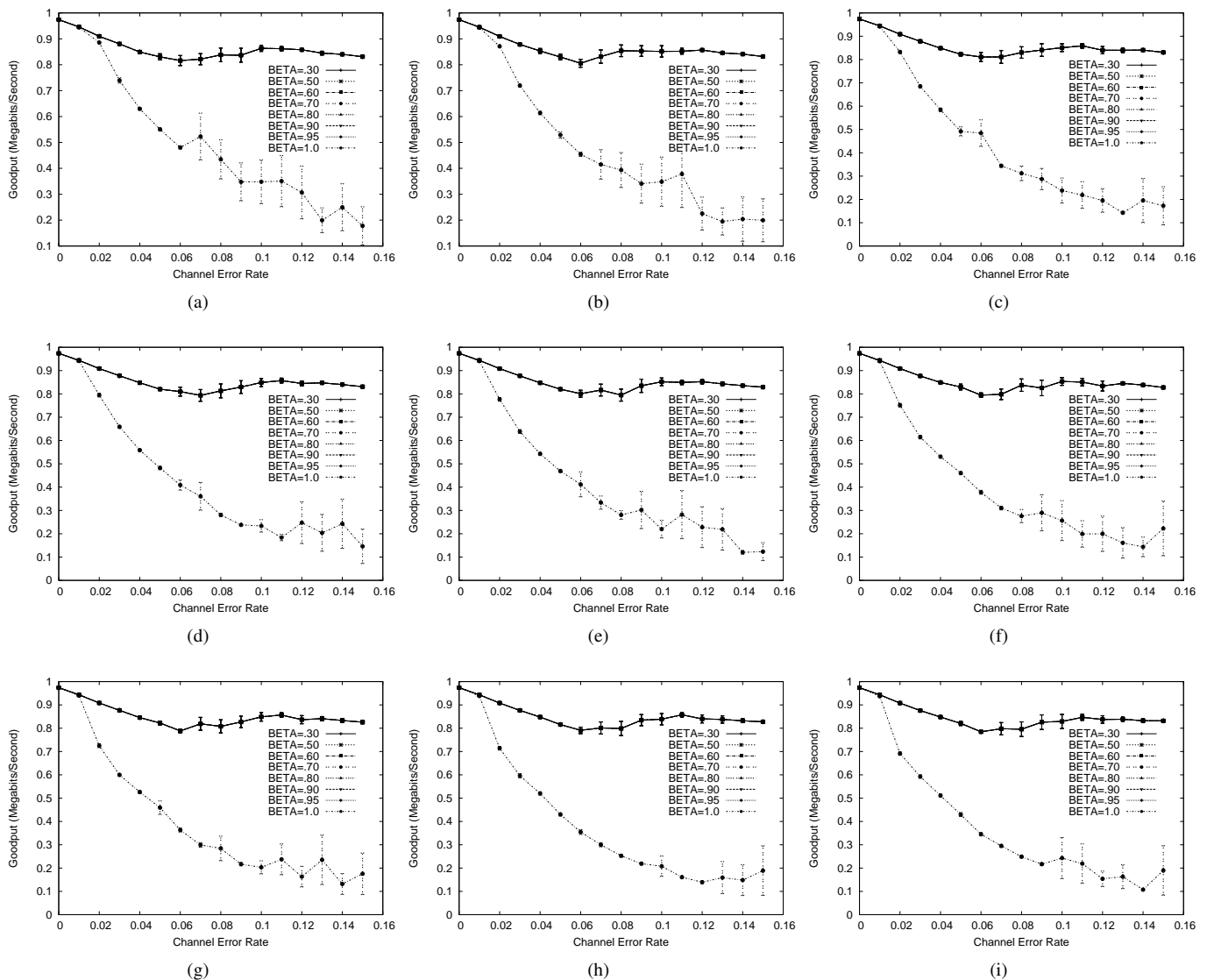


Fig. 5. Goodput performance of TCP-NCL against MRRL under infrastructure-based wireless network: (a) MRRL = 250. (b) MRRL = 300. (c) MRRL = 400. (d) MRRL = 500. (e) MRRL = 600. (f) MRRL = 700. (g) MRRL = 800. (h) MRRL = 900. (i) MRRL = 1000.

TCP-NCL-RTT-Update is partially verified as a robust RTT sampling mechanism. On the other hand, among all the solutions for packet reordering, TCP-DCR gets the second worst performance gain due to its inability of maintaining accurate RTT sampling with *Type II Reordering*.

V. CONCLUSIONS

In this paper, we have adapted TCP to tackle packet reordering and random packet loss over wireless networks. A temporal approach, known as TCP for non-congestion loss (TCP-NCL), has been proposed. The packet retransmission decision (PRD) model is constructed to provide an analytical approach for optimizing the waiting time before triggering a packet retransmission. Three algorithms, TCP-NCL-RT, TCP-NCL-SRDT, and TCP-NCL-RTT-Update, have been developed within the framework of TCP-NCL.

Various test cases have been designed to simulate the prescribed problems of *Type I Reordering*, *Type II Reordering*, and random packet loss. The performance of TCP-NCL under these cases have been examined against various combinations of values for its preset parameters and compared with other TCP variants. Our simulation results reveal that TCP-NCL demonstrates robustness against variations of its preset parameters and that it offers the best performance when compared with other TCP variants. Moreover, TCP-NCL is verified to fulfill its designated functions. Further simulation experiments are currently in progress to investigate the internal behavior of TCP-NCL as well as other important performance issues, including TCP-friendliness.

There are several possible extensions to our work, some of which are listed below:

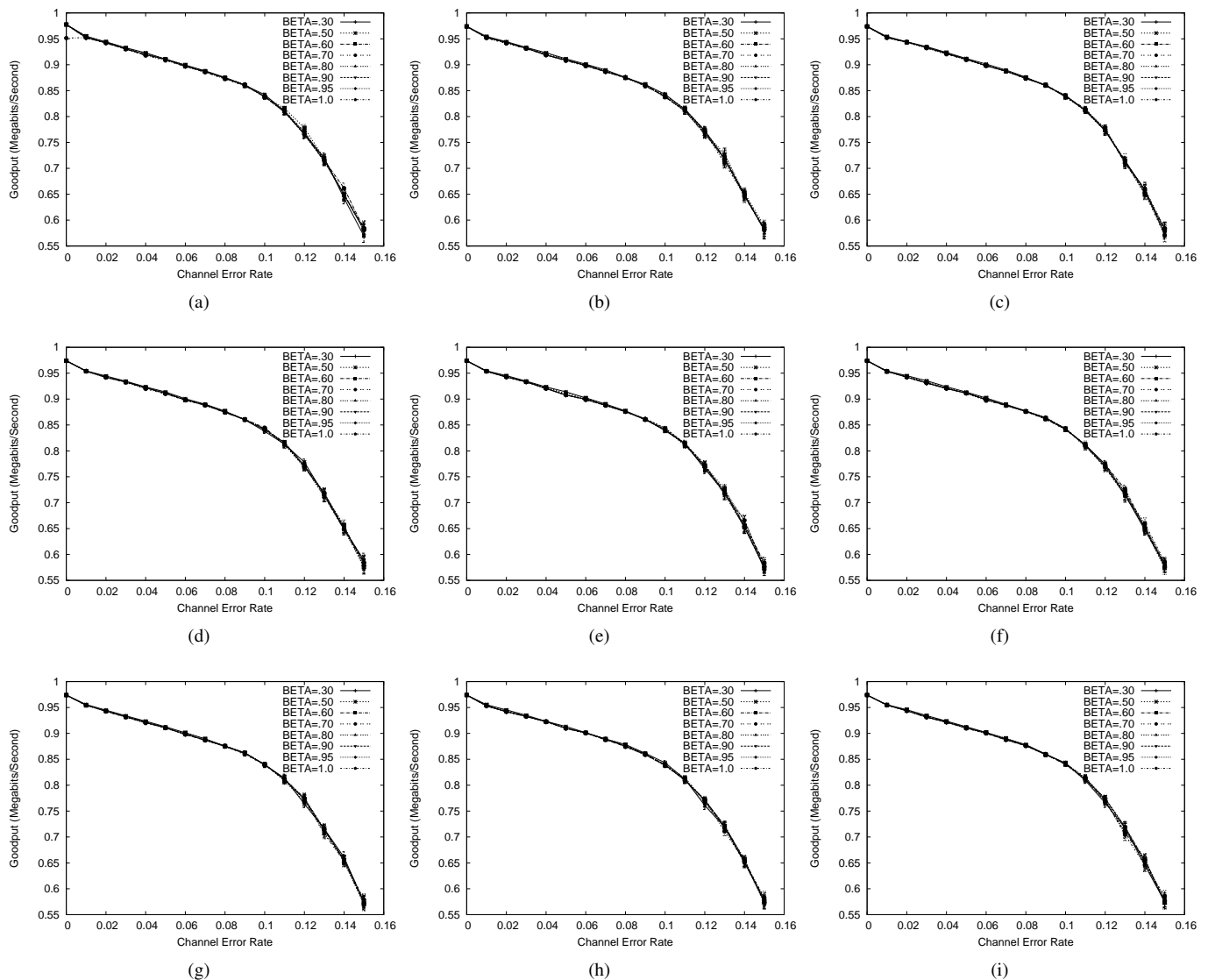


Fig. 6. Goodput performance of TCP-NCL against MRRL under multi-hop wireless network: (a) MRRL = 250. (b) MRRL = 300. (c) MRRL = 400. (d) MRRL = 500. (e) MRRL = 600. (f) MRRL = 700. (g) MRRL = 800. (h) MRRL = 900. (i) MRRL = 1000.

- 1) improve the computational complexity of TCP-NCL algorithms, especially that of TCP-NCL-RTT-Update;
- 2) develop a distribution model of RTT to make TCP-NCL a memoryless algorithm, and;
- 3) implement and examine the performance of TCP-NCL on experimental testbeds.

REFERENCES

- [1] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. *Request for Comments*, RFC 2581, Network Working Group, Internet Engineering Task Force, April 1999.
- [2] H. Balakrishnan, V.N. Padmanabhan, S. Seshan, and R.H. Katz. A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. *IEEE/ACM Transactions on Networking*, Vol. 5, No. 6, pp. 756-769, December 1997.
- [3] S. Bhandarkar and A.L.N. Reddy. TCP-DCR: Making TCP Robust to Non-Congestion Events. *Lecture Notes in Computer Science*, Vol. 3042, pp. 712-724, May 2004.
- [4] S. Bohacek, J.P. Hespanha, J. Lee, C. Lim, and K. Obraczka. A New TCP for Persistent Packet Reordering. *IEEE/ACM Transactions on Networking*, Vol. 14, No. 2, pp. 369-382, April 2006.
- [5] C. Casetti, M. Gerla, S. Mascolo, M.Y. Sanadidi, and R. Wang. TCP Westwood: End-to-End Congestion Control for Wired/Wireless Networks. *Wireless Networks*, Vol. 8, No. 5, pp. 467-479, 2002.
- [6] K. Fall and K. Varadhan. The *ns* Manual (formerly *ns* Notes and Documentation). *The VINT Project*, 16 September 2006.
- [7] C.P. Fu and S.C. Liew. TCP VenO: TCP enhancement for transmission over wireless access networks. *IEEE Journal on Selected Areas in Communications*, Vol. 21, No. 2, pp. 216-228, February 2003.
- [8] F. Hu and N. Sharma. Enhancing Wireless Internet Performance. *IEEE Communications Surveys and Tutorials*, Vol. 4, No. 1, pp. 2-15, December 2002.
- [9] P. Karn and C. Partridge. Improving Round-Trip Time Estimates in Reliable Transport Protocols. *ACM Transactions on Computer Systems*, Vol. 9, No. 4, pp. 364-373, November 1991.

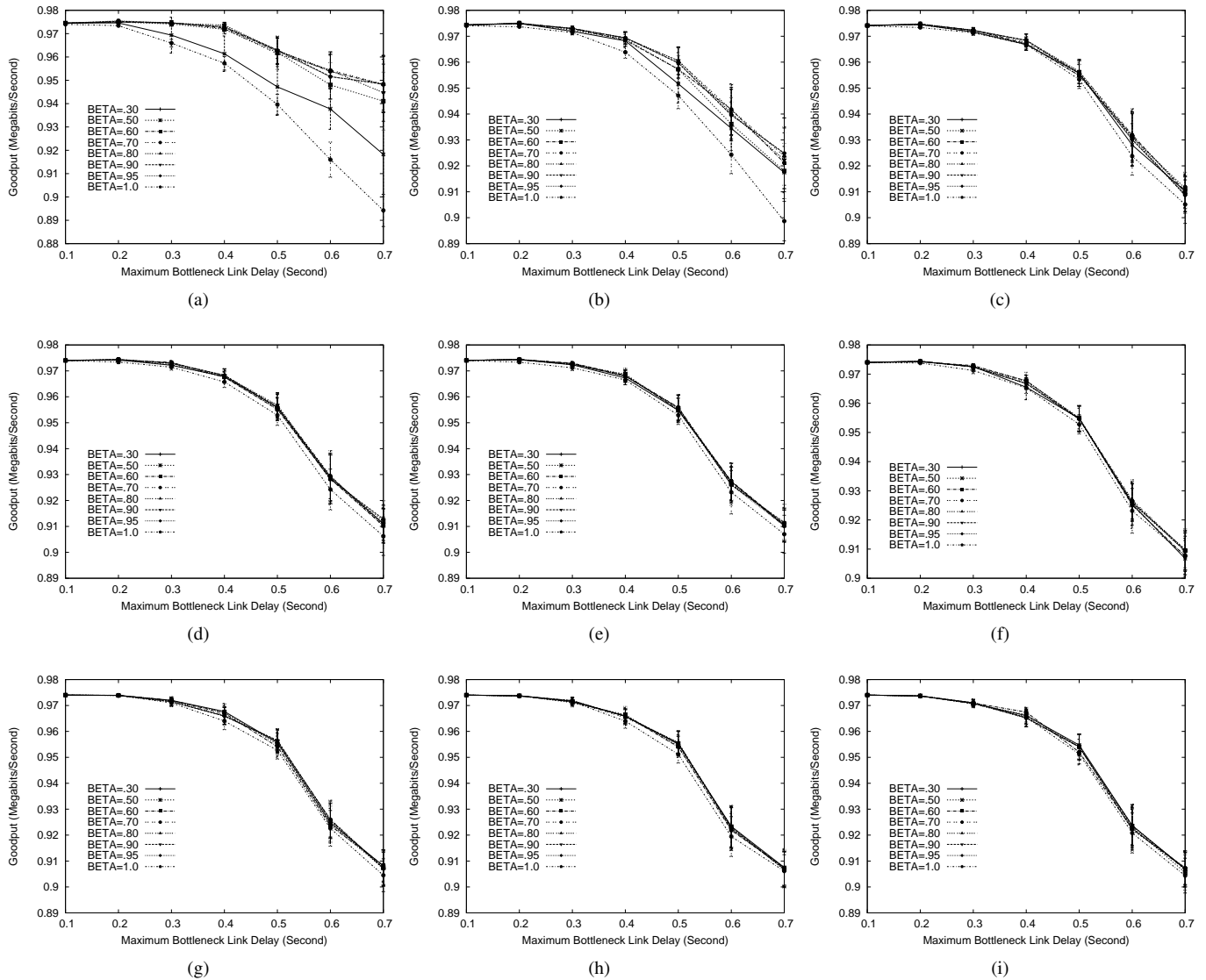
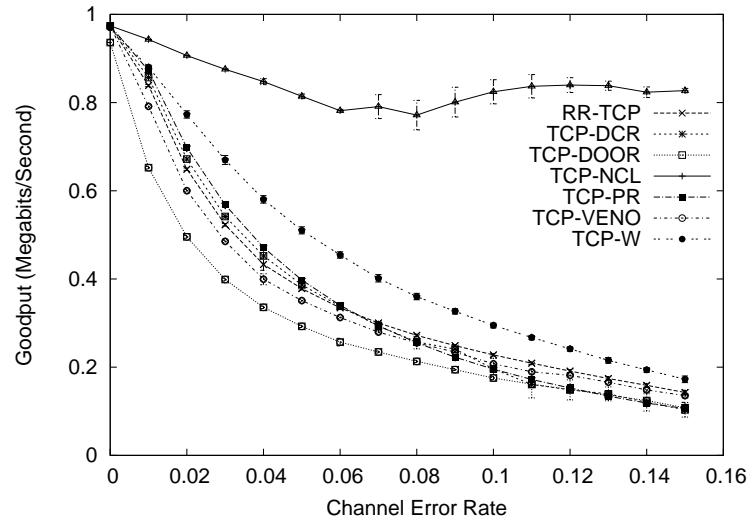
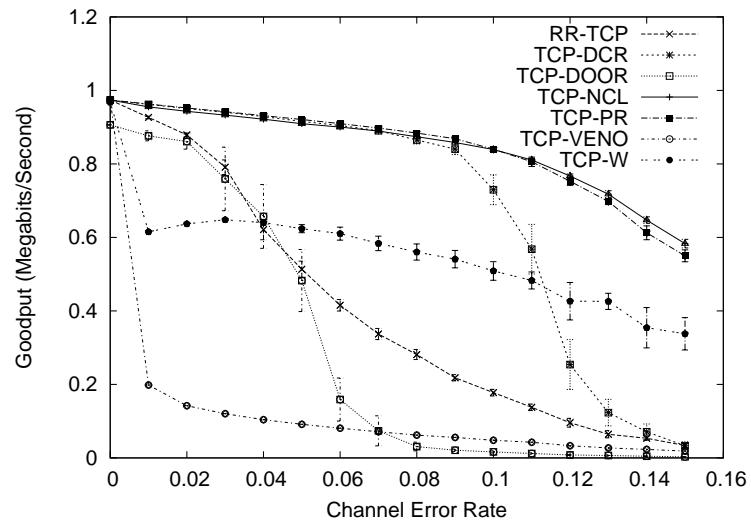


Fig. 7. Goodput performance of TCP-NCL against MRRL under wired network with a bottleneck link: (a) MRRL = 250. (b) MRRL = 300. (c) MRRL = 400. (d) MRRL = 500. (e) MRRL = 600. (f) MRRL = 700. (g) MRRL = 800. (h) MRRL = 900. (i) MRRL = 1000.

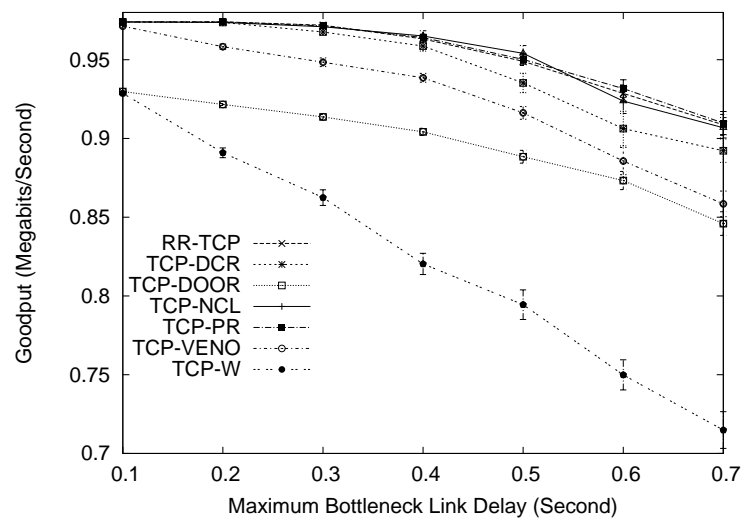
- [10] K.-C. Leung and V.O.K. Li. Transmission Control Protocol (TCP) in Wireless Networks: Issues, Approaches, and Challenges. *IEEE Communications Surveys and Tutorials*, Vol. 8, No. 4, pp. 64-79, Fourth Quarter 2006.
- [11] K.-C. Leung, V.O.K. Li, and D. Yang. An Overview of Packet Reordering in Transmission Control Protocol (TCP): Problems, Solutions, and Challenges. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 18, No. 4, pp. 522-535, April 2007
- [12] J. Liu and S. Singh. ATCP: TCP for Mobile Ad Hoc Networks. *IEEE Journal on Selected Areas in Communications*, Vol. 19, No. 7, pp. 1300-1315, July 2001.
- [13] D. Mitzel. Overview of 2000 IAB Wireless Internetworking Workshop. *Request for Comments, RFC 3002, Network Working Group*, December 2000.
- [14] J. Postel. Transmission Control Protocol. *Request for Comments, RFC 793, Protocol Specification, DARPA Internet Program*, September 1981.
- [15] A.K. Singh and K. Kankipati. TCP-ADA: TCP with Adaptive Delayed Acknowledgement for Mobile Ad Hoc Networks. *Proceedings of IEEE WCNC 2004*, Vol. 3, pp. 1685-1690, Atlanta, GA, USA, 21-25 March 2004
- [16] F. Wang and Y. Zhang. Improving TCP Performance over Mobile Ad-Hoc Networks with Out-Of-Order Detection and Response. *Proceedings of ACM MOBIHOC 2002*, pp. 217-225, Lausanne, Switzerland, 9-11 June 2002.
- [17] D. Yang, K.-C. Leung, and V.O.K. Li. Simulation-Based Comparisons of Solutions for TCP Packet Reordering in Wireless Networks. *Proceedings of IEEE WCNC 2007*, pp. 3238-3243, Hong Kong, China, 11-15 March 2007.
- [18] M. Zhang, B. Karp, S. Floyd, and L. Peterson. RR-TCP: A Reordering-Robust TCP with DSACK. *Proceedings of IEEE ICNP 2003*, pp. 95-106, Atlanta, GA, USA, 4-7 November 2003.



(a)



(b)



(c)

Fig. 8. Goodput performance under various topologies: (a) Infrastructure-based wireless network. (b) Multi-hop wireless network. (c) Wired network with a bottleneck link.